

EE441 PA1

```

xpark@conelge MSYS /c/Users/xpark/OneDrive/Documents/METU/TERM7/EE441/EE441_PA1_2375426_P1
• $ ./EE441*
Contents of List1: 0 1 2 3 4 5 6 7 8 9
Contents of List2: 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
index_test[5] for List1: 5
index_test[5] for List2: 50
Contents of List1 After Inserting 1.234: 0 1 1.234 2 3 4 5 6 7 8 9
Contents of List2 After Inserting 87.42: 25 30 35 40 45 50 55 60 65 70 75 80 85 87.42 90 95 100
Contents of List1 After Removing [7]: 0 1 1.234 2 3 4 5 7 8 9
Contents of List2 After Removing [3]: 25 30 35 45 50 55 60 65 70 75 80 85 87.42 90 95 100
Find_test with 1.234 in List1: 2
Find_test with 87.42 in List2: 12
Copy List2 to List1:
List1: 25 30 35 45 50 55 60 65 70 75 80 85 87.42 90 95 100
List2: 25 30 35 45 50 55 60 65 70 75 80 85 87.42 90 95 100

=== Additional Tests ===
Caught exception when trying to insert into a full list: List Length is Maximum Already
Caught exception when attempting to remove from an empty list: You Have Entered Out of Range Index, Enter Again
Caught exception when searching for a non-existent number: Number not found in the list.
Contents of an empty list: The list is empty.
Accessing index 0 of an empty list: Caught exception when trying to access an index in an empty list: You Have Entered Out of Range Index, Enter Again

```

Figure 1: Terminal Output of the Program

Question 6. Time Complexity of the insert Function:

In the SortedList class, the insert function currently employs a linear search to determine the appropriate position for inserting a new number. This process consists of two key components:

1. Linear Search

- In the worst-case scenario, if the new number being added is either greater than all existing numbers or less than all of them, the function must traverse the entire list to identify the correct insertion point. Consequently, if there are n elements in the list, the linear search requires $O(n)$ time.

2. Shifting Elements

- After locating the correct position, the function must move all subsequent elements one position to the right to make room for the new number. In the worst case, this could involve shifting all n elements. This shifting operation also takes $O(n)$ time.

Putting these two parts together, the overall time complexity of the insert function is $O(n) + O(n) = O(n)$.

If binary search were utilized instead of a linear search, the complexity for the search component would decrease to $O(\log n)$ since binary search works by repeatedly dividing the list in half. However, since the shifting operation still requires $O(n)$ time in the worst case, the overall complexity would still be $O(\log n) + O(n) = O(n)$.

Question 8. Time Complexity of the remove Function:

In contrast to the insert function, the remove function does not involve a search operation. Therefore, we will only focus on the shifting of elements

1. Shifting Elements

- After an element is removed, all subsequent elements need to be shifted to fill the gap left by the removed element. In the worst-case scenario, if the element to be removed is the first one, the function must shift $n-1$ elements, which results in $O(n)$ time complexity.

As a result, the overall time complexity for the remove function is $O(n)$.

Question 10. Time Complexity of the find Function:

The find function exclusively utilizes binary search. This method continually divides the list in half until it either locates the target number or exhausts the options available. The efficiency of binary search allows it to perform the search in $O(\log n)$ time when operating on a sorted list. Therefore, the time complexity for the find function is $O(\log n)$.